ГЛАВА XX.
Ансамбли алгоритмов

Таланты выигрывают матчи,
а командная работа и ум – чемпионаты
М. Джордан

Комитет – это двенадцать человек,
делающих работу одного
Д. Ф. Кеннеди

Ансамбли алгоритмов: примеры и обоснование

Ансамблем (Ensemble, Multiple Classifier System) называется алгоритм, который состоит из нескольких алгоритмов машинного обучения[1], а процесс построения ансамбля называется **ансамблированием (ensemble learning)**. Простейший пример ансамбля в регрессии – усреднение нескольких регрессоров[2]:

$$a(x) = \frac{1}{n}\big(b_1(x) + \ldots + b_n(x)\big). \qquad \text{(XX.1)}$$

Алгоритмы, из которых состоит ансамбль (в (XX.1) это b_t, $t \in \{1,2,\ldots,n\}$), называются **базовыми алгоритмами (base learners).** Если рассматривать значения базовых алгоритмов на объекте x как независимые случайные величины $\xi_1,\ldots,\xi_n$ с одинаковым матожиданием y и одинаковой конечной дисперсией σ^2, то понятно, что случайная величина (XX.1) имеет такое же матожидание, но меньшую дисперсию:

$$\xi = \frac{1}{n}\big(\xi_1 + \ldots + \xi_n\big),$$

$$\mathbf{E}\xi = \frac{1}{n}\big(\mathbf{E}\xi_1 + \ldots + \mathbf{E}\xi_n\big) = \mathbf{E}\xi_i = y,$$

[1] Использует результаты работы нескольких алгоритмов.

[2] Раньше через n обозначали число признаков, в этой главе число признаков не фигурирует, поэтому коллизии не будет.

$$\mathbf{D}\xi = \frac{1}{n^2}\left(\mathbf{D}\xi_1 + \ldots + \mathbf{D}\xi_n\right) = \frac{\mathbf{D}\xi_i}{n} = \frac{\sigma^2}{n}.$$

Требование равенства матожиданий ответов базовых алгоритмов на конкретном объекте вполне естественное: если мы берём алгоритмы из несмещённой модели

$$\mathbf{E}b_i(x) = y(x),$$

то их матожидания не только совпадают, но и равны значению истинной регрессионной метки. А вот у требование равенства дисперсий мы сделали для удобства[1].

В задачах классификации простейший пример ансамбля – **комитет большинства**:

$$a(x) = \text{mode}\left(b_1(x),\ldots,b_n(x)\right), \qquad \text{(XX.2)}$$

где mode – мода (значение, которое встречается чаще других среди аргументов функции). Если рассмотреть задачу классификации с двумя классами $\{0,1\}$ и три бинарных классификатора, каждый из которых ошибается с вероятностью p, то в предположении, что их ответы являются независимыми случайными величинами, получаем, что их комитет большинства ошибается с вероятностью

$$p_3 = 3\cdot p\cdot p\cdot(1-p) + p\cdot p\cdot p = p^2(3-2p),$$

здесь с вероятностью $p\cdot p\cdot(1-p)$ ошибаются первый и второй при верном ответе у третьего, с вероятностью p^3 ошибаются все три. Как видно на рис. XX.1, это выражение может быть существенно меньше p (при $p=0.2$ почти в два раза), т.е. использование такого ансамбля уменьшает ошибку базовых алгоритмов.

Если рассмотреть большее число алгоритмов, то по неравенству Хёфдинга[2] (Hoeffding) ошибка комитета большинства

$$\sum_{t=0}^{\lfloor n/2\rfloor} C_n^t (1-p)^t p^{n-t} \le e^{-\frac{1}{2}n(2p-1)^2},$$

[1] В задачах рассмотрим более общий случай.

[2] W. Hoeffding. Probability inequalities for sums of bounded random variables // Journal of the American Statistical Association. — 1963. — Т. 58, № 301. — С. 13–30.

т.е. экспоненциально убывает с ростом числа базовых алгоритмов.

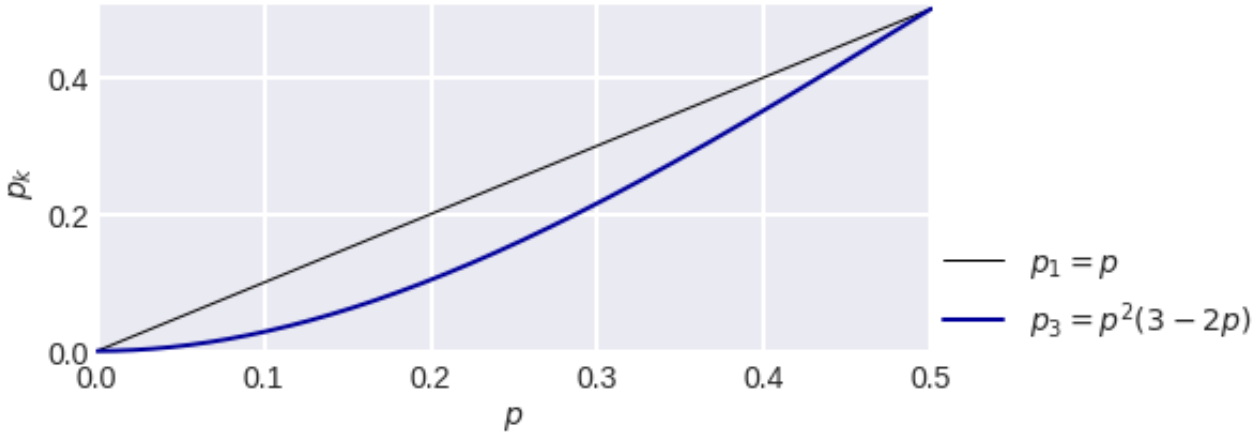

Рис. XX.1. График вероятности ошибки комитета большинства.

Наше теоретическое обоснование не совсем годится для реальной практической ситуации, поскольку **регрессоры / классификаторы не являются независимыми**:

- они решают одну задачу (вся входная информация совпадают),
- настраиваются на один целевой вектор (ответы всех хороших регрессоров, скорее всего, коррелируют с целевыми значениями),
- могут быть из одной модели (или из нескольких, но всё равно небольшого числа).

Поэтому большинство приёмов в прикладном ансамблировании направлено на то, чтобы ансамбль был «достаточно разнообразным», тогда **ошибки одних базовых алгоритмов на отдельных объектах будут компенсироваться правильными ответами других алгоритмов**. По сути, при построении ансамбля есть две цели:

Основной принцип ансамблирования – обеспечить разнообразие базовых алгоритмов.

- достаточно высокое качество базовых алгоритмов,
- высокое разнообразие (diversity) базовых алгоритмов.

Приведём такую иллюстрацию пользы разнообразия базовых алгоритмов в комитете большинства. Будем бинарным вектором кодировать правильные ответы алгоритма на контрольной выборке в задаче с двумя классами: 1 – правильный ответ, 0 – ошибка. Если мы построим несколько одинаковых алгоритмов (по крайней мере, в смысле совпадения указанных

характеристических векторов правильных ответов), то голосование по этим алгоритмам будет иметь точно такой же вектор:

алгоритм 1: 1111110000 качество = 0.6
алгоритм 2: 1111110000 качество = 0.6
алгоритм 3: 1111110000 качество = 0.6
ансамбль : 1111110000 качество = 0.6

Теперь построим похожие алгоритмы (правильность/неправильность на многих объектах совпадают):

алгоритм 1: 1111110000 качество = 0.6
алгоритм 2: 1111101000 качество = 0.6
алгоритм 3: 1111100100 качество = 0.6
ансамбль : 1111100000 качество = 0.5

Построим совершенно разные алгоритмы (здесь можно случайно раскидать по позициям единицы и нули в каждом векторе):

алгоритм 1: 1010010111 качество = 0.6
алгоритм 2: 1100110011 качество = 0.6
алгоритм 3: 1111110000 качество = 0.6
ансамбль : 1110110011 качество = 0.7

Формально получилось, что качество комитета большинства в последнем случае самое высокое: ансамбль «разнообразных» алгоритмов показал лучшее качество[1].

Разнообразие можно повысить по-разному обучая базовые алгоритмы. На рис. XX.2 схематично показаны преобразования, которые осуществляют для повышения разнообразия: выделены элементы, которые можно «варьировать» (изменять при обучении). Перечислим сущности варьирования с указанием техник ансамблирования, которые они порождают (далее рассмотрим их подробнее):

Если нарисовать, как решается задача, то из рисунка понятно, какие компоненты решения можно варьировать.

- обучающая выборка[2] (бэгинг),
- признаки (случайные подпространства),
- целевой вектор (перекодировка или деформации целевого признака),
- модели (блендинг, стэкинг),

[1] Здесь есть, правда, некоторое лукавство (см. задачи).

[2] Имеется в виду, что разные базовые алгоритмы обучаются на разных обучающих выборках.

- алгоритмы в модели (использование разных алгоритмов в рамках одной[1], рандомизация в алгоритме – случайный лес).

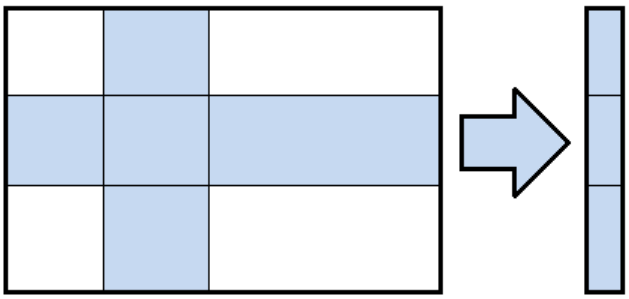

Рис. XX.2. Символическое изображение матрицы данных, алгоритма (преобразования) и вектора целевых значений.

Обоснования использования ансамблей в машинном обучении обычно бывают **статистическими (statistical)** – их привели выше, когда мы оценивали ошибку ансамбля исходя из вероятностной природы ответов базовых алгоритмов, **вычислительными (computational)** – поскольку обучение это решение задачи оптимизации, то ансамбль распараллеливает процесс: мы параллельно обучаем несколько базовых алгоритмов и на их основе пытаемся получить более качественное решение, **функциональными (representational)** – часто ансамблем можно представить более сложную функцию, чем любым базовым алгоритмом. Последнее можно проиллюстрировать рис. XX.3: показана задача бинарной классификации, в которой нельзя получить безошибочный линейный классификатор, и два линейных классификатора. Третий можно выбрать, например, относящим все объекты в класс 0. Комитет большинства трёх таких линейных классификаторов безошибочно классифицирует выборку, его разделяющая поверхность кусочно-линейная.

Ансамбль, как правило, сложнее базовых алгоритмов.

Отметим, что на рис. XX.3 виден также следующий забавный эффект: самая красная треугольная область соответствует максимальной уверенности ансамбля в том, что там лежат объекты класса 0: три базовых алгоритма из трёх голосуют за это. При этом в этой области практически нет обучающих объектов!

Не всегда найденные паттерны соответствуют данным.

[1] Например, усреднение алгоритмов, которые являются представителями одной модели с разными значениями гиперпараметров.

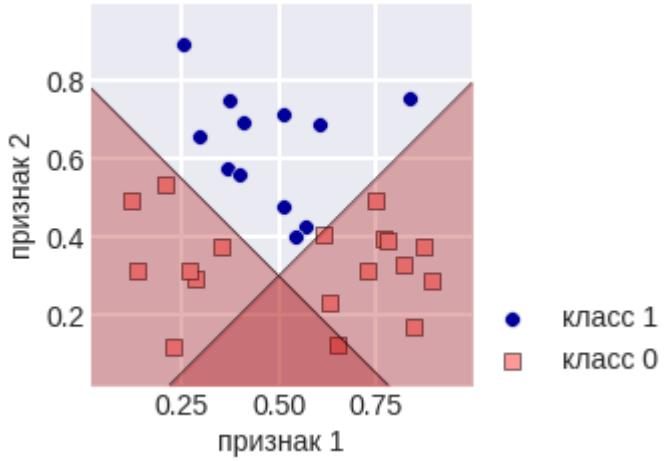

Рис. XX.3. Линейные классификаторы в задаче с нелинейной природой данных.

Усреднение алгоритмов, комитеты, голосование (Voting Ensembles)

В общем случае ансамбль можно записать в виде

$$a(x) = b\big(b_1(x),\ldots,b_n(x)\big),$$

где b – некоторая функция или, как увидим дальше, алгоритм, который принято называть **мета-алгоритмом** (**meta-estimator**). Как видим, выражения (XX.1) и (XX.2) являются частным случаем этой записи. В регрессии в качестве b можно выбрать среднее арифметическое. В общем случае **может использоваться любое усреднение по Коши**[1]. Например, в **задачах обнаружения аномалий** в качестве мета-классификатора **может использоваться максимум**: каждый базовый алгоритм выдаёт свою оценку вероятности того, что конкретный объект является аномалией (не похож на обучающую выборку), мы полагаемся на максимальную оценку. Поскольку такие задачи возникают в областях, где нельзя пропустить аномалию (обнаружение поломок, хакерских атак, инсайдеров на бирже и т.п.), действуют по логике «поднимать тревогу при малейшем подозрении».

Часто при использовании функции ошибки MAE[2] в задаче регрессии **медиана базовых алгоритмов** показывает качество выше, чем среднее арифметическое. В задачах бинарной классификации при использовании показателя качества AUC ROC – часто усредняют не оценки принадлежности классу 1, а их ранки[3]:

[1] Средним по Коши называется любая функция $f:\ \min(z_1,\ldots,z_n) \le f(z_1,\ldots,z_n) \le \max(z_1,\ldots,z_n)$.

[2] См. главу «Функции ошибки».

[3] Ранк – порядковый номер при упорядочивании по неубыванию, $\text{rank}:[0.3,\ 0.1,\ 0.9] \to [1,\ 0,\ 2]$.

$$a(x) = \frac{1}{n}\big(\text{rank}(b_1(x)) + \ldots + \text{rank}(b_n(x))\big).$$

Можно это попробовать объяснить тем, что для площади под ROC-кривой важно «правильное упорядочивание» объектов в задаче бинарной классификации, а не конкретные значения оценок. Функция rank позволяет в меньшей степени учитывать сами значения, которые выдают алгоритмы, а в большей – порядок на объектах контрольной выборки.

В общем случае, ансамбль часто конструируют под определённый функционал качества в виде

$$a(x) = g\left(\frac{1}{n}\big(f(b_1(x)) + \ldots + f(b_n(x))\big)\right)$$

(эта формула обобщает «среднее по Колмогорову[1]»). Вместо обычного усреднения / голосования часто используют **усреднение с весами (weighted averaging)**:

$$a(x) = \frac{1}{w_1 + \ldots + w_n}\big(w_1 \cdot b_1(x) + \ldots + w_n \cdot b_n(x)\big),$$

и **голосование с весами**:

$$a(x) = \arg\max_j \left[\sum_{t:b_t(x)=j} w_t\right],$$

когда есть основания по-разному доверять разным алгоритмам, участвующим в ансамбле. Например, в соревновании Netflix Prize[2] была предложена техника **Feature-Weighted Linear Stacking**, в которой веса базовых алгоритмов являются линейными регрессиями:

$$w_t(x) = \sum_i w_{ti} f_i(x),$$

где $f_i(x)$ – значение i-го признака объекта x (суммирование идёт по всем признакам). Тогда ансамбль запишется в виде

$$a(x) = w_1(x) \cdot b_1(x) + \ldots + w_n(x) \cdot b_n(x) =$$

[1] Среднее по Колмогорову – $f^{-1}\big((f(z_1)+\ldots+f(z_n))/n\big)$.

[2] https://en.wikipedia.org/wiki/Netflix_Prize

$$= \sum_t (\sum_i w_{ti} f_i(x)) b_t(x) = \sum_{t,i} w_{ti} b_t(x) f_i(x),$$

в последней формуле суммирование ведётся по базовым алгоритмам и признакам. Поэтому ансамбль превращается в линейную регрессию над признаками вида $b_t(x) f_i(x)$, т.е. над произведениями базовых алгоритмов и исходных признаков.

Более сложные ансамбли, в которых мета-алгоритм может быть алгоритмом машинного обучения (классификатором или регрессором), рассмотрим в главе про стекинг.

Бэгинг

Идея **бэгинга (b**ootstrap **agg**regat**ing)** проста: каждый базовый алгоритм обучается на случайном подмножестве обучающей выборки, причём подвыборка формируется с помощью бутстрепа. В этом случае, даже используя одну модель алгоритмов, мы получаем различные базовые алгоритмы. Вообще, есть следующие модификации этой идеи:

Метод	Описание
Бэгинг[1]	Подвыборка обучающей выборки берётся с помощью бутстрепа – выбора с возвращением
Пэстинг (Pasting[2]**)**	Используется обычная случайная подвыборка обучающей выборки – выбор без возвращения
Случайные подпространства (Random Subspaces[3]**)**	Берётся случайное подмножество признаков
Случайные патчи (Random Patches[4]**)**	Одновременно берётся случайное подмножество объектов и признаков
CV-комитеты (cross-validated committees[5]**)**	Выборка разбивается на k фолдов, базовые алгоритмы получаются обучением на $(k-1)$-м фолде

[1] Breiman L. Bagging predictors //Machine learning. – 1996. – Т. 24. – С. 123-140.

[2] Breiman L. Pasting small votes for classification in large databases and on-line //Machine learning. – 1999. – Т. 36. – С. 85-103.

[3] Ho T. K. The random subspace method for constructing decision forests //IEEE transactions on pattern analysis and machine intelligence. – 1998. – Т. 20. – №. 8. – С. 832-844.

[4] Louppe G., Geurts P. Ensembles on random patches //Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2012, Bristol, UK, September 24-28, 2012. Proceedings, Part I 23. – Springer Berlin Heidelberg, 2012. – С. 346-361.

[5] Parmanto B., Munro P. W., Doyle H. R. Reducing variance of committee prediction with resampling techniques //Connection Science. – 1996. – Т. 8. – №. 3-4. – С. 405-426.

Вероятность быть отобранным при бутстрепе (для достаточно больших выборок):

$$1-\frac{1}{e}\approx 0.632,$$

т.е. примерно 36.8% объектов оказываются «за бортом», такие объекты называются **out-of-bag-наблюдениями (OOB).**

Алгоритм Бэгинга

1. Цикл по t (номер базового алгоритма).
   1.1. Взять подвыборку $[X', y']$ обучающей выборки $[X, y]$ с помощью бутстрепа.
   1.2. Обучить t-й базовый алгоритм на этой подвыборке:
   $$b_t = \text{fit}(X', y')$$
2. Ансамбль строится усреднением базовых алгоритмов, например, для задач регрессии:
   $$a(x) = \frac{1}{n}\left(b_1(x) + \ldots + b_n(x)\right).$$

Важно понимать, что подвыборки обучающей выборки берутся в бэгинге для того, чтобы базовые алгоритмы получались непохожими. Есть модели алгоритмов, которые **устойчивы (stable learners)** относительно взятия подвыборок (kNN при k>3, SVM), их не следует использовать в бэгинге. В бэгинге, как правило, используются базовые алгоритмы с большим разбросом (high variance, см. главу «Сложность»), усреднение уменьшает разброс, при этом следует использовать несмещённые алгоритмы (small bias), это гарантирует несмещённость усреднения.

Бэгинг – для несмещённых базовых алгоритмов с большим разбросом.

На рис. XX.4 показано несколько алгоритмов (первые три иллюстрации в каждом ряду) и результат бэгинга над 100 алгоритмов рассматриваемой модели (последняя иллюстрация в ряду). Были использованы пеньки (деревья глубины 1) – у них большое смещение и деревья большой глубины – у них маленькое смещение. Результат каждого алгоритма показан на фоне выборки, на которой он обучался. Бэгинг над пеньками в итоге не настроился на обучающую выборку, в отличие от бэгинга над глубокими деревьями.

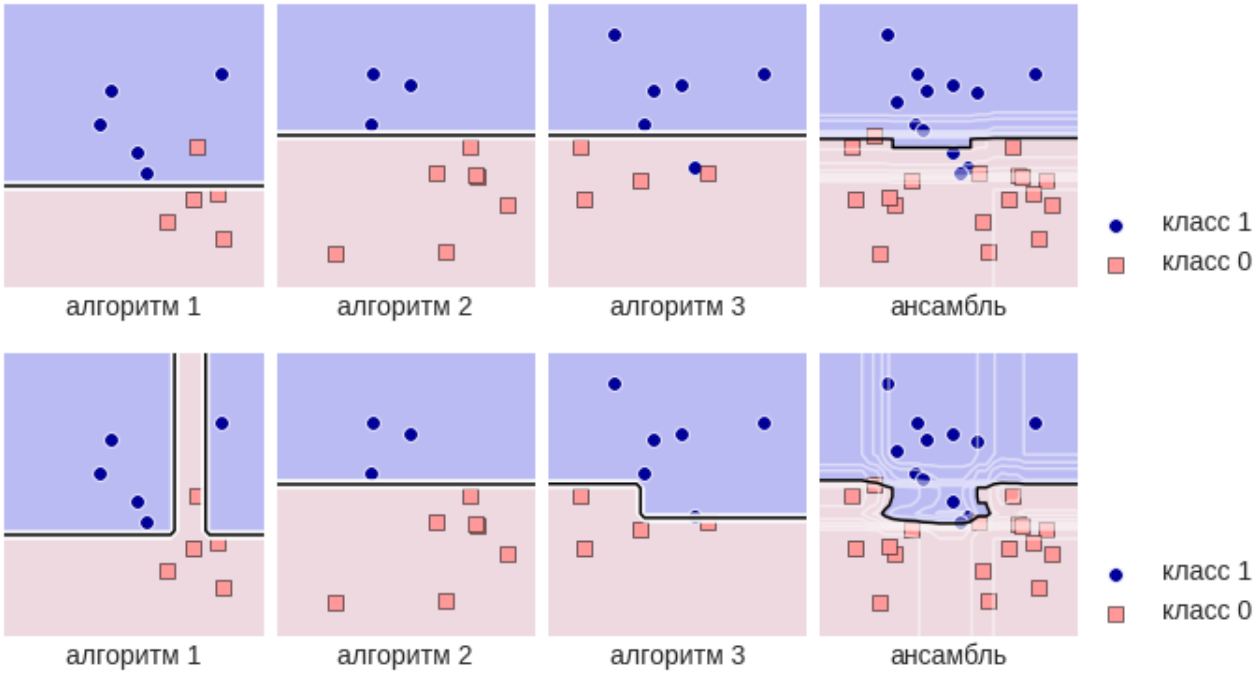

Рис. XX.4. Три первых алгоритма (верхний ряд – деревья глубины 1, нижний – деревья бесконечной глубины) и результат бэгинга.

На рис. XX.5 показаны результаты алгоритмов разных моделей и бэгинга над ними. Заметим, что для стабильных моделей (SVM, логистическая регрессия) совсем не заметен эффект ансамблирования (результаты отдельного алгоритма и ансамбля практически совпадают), а для нестабильных (5NN, решающее дерево) ансамбль «сглаживает» решение, практически убирая артефакты, связанные с выбросами.

Бэгинг позволяет получать т.н. **out-of-bag(OOB)-ответы** модели. Идея очень простая: каждый базовый алгоритм обучается на подвыборке, в которую, вообще говоря, попадают не все объекты из обучения, поэтому на остальных объектах (OOB) можно узнать ответы алгоритма. Эти ответы можно усреднить следующим образом:

$$a_{\text{OOB}}(x_j) = \frac{1}{|\{i : x_j \in \text{OOB}_i\}|} \sum_{i:\, x_j \in \text{OOB}_i} b_i(x_j),$$

здесь OOB_i – out-of-bag подвыборка для i-го базового алгоритма, суммирование производится по номерам алгоритмов, для которых j-й объект попал в out-of-bag. При достаточном числе базовых алгоритмов в бэгинге таким образом оценивается ответ на всех объектах обучения. Причём это «честный» ответ: те алгоритмы, которые участвовали в его формировании «не видели» истинной метки соответствующего объекта.

Аналогично оценивается **out-of-bag(OOB)-ошибка** бэгинга. Можно оценить ошибку с помощью полученных OOB-ответов, но чаще делают проще: для каждого базового алгоритма ошибку оценивают на объектах, не попавших в его обучение, а затем ошибки усредняют.

Обучение не на всех данных даёт способы параллельной оценки качества.

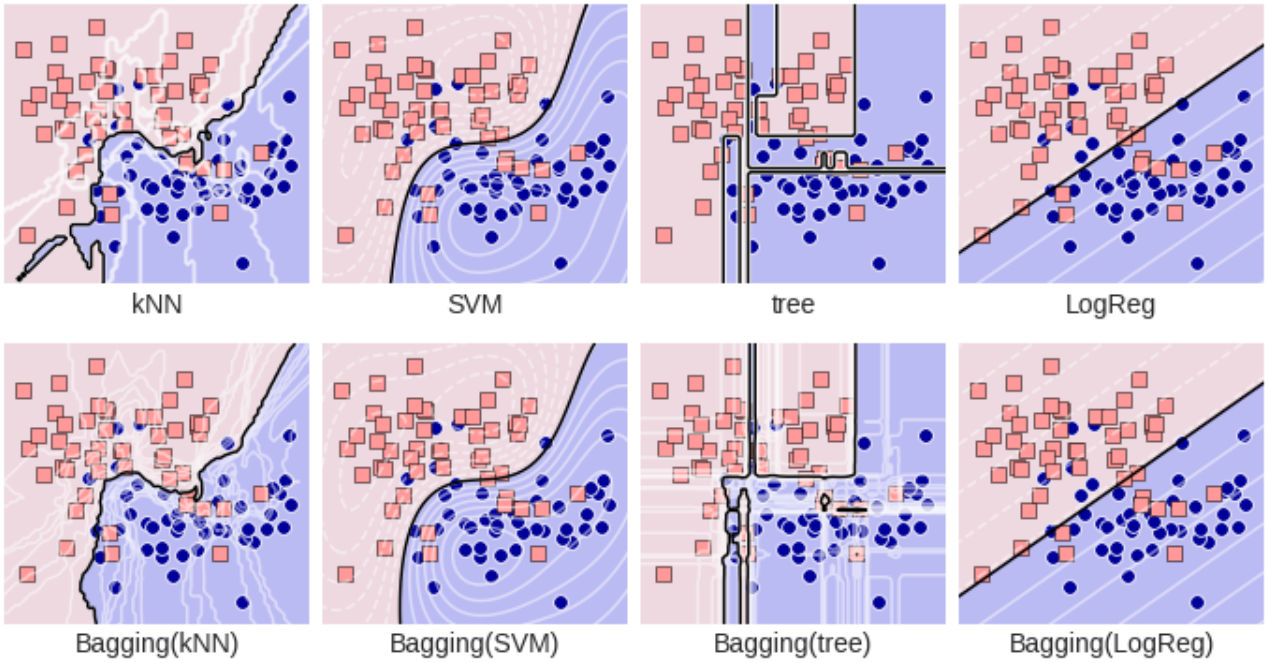

Рис. XX.5. Разделение выборки отдельными моделями (верхний ряд) и бэгингом над алгоритмами этих моделей (нижний).

Про важный частный случай бэгинга над решающими деревьями – случайный лес поговорим в отдельной главе.

Бустинг

Главная идея бустинга – базовые алгоритмы строятся не независимо, каждый следующий мы строим так, чтобы он исправлял ошибки предыдущих и (потенциально) повышал качество всего ансамбля. Первый успешный вариант бустинга – **Адаптивный бустинг AdaBoost** (Adaptive Boosting), его мы подробно рассмотрим ниже. Основной принцип реализации бустинга – **Forward stagewise additive modeling (FSAM)**. Пусть, например, решается задача регрессии с обучающей выборкой $(x_i, y_i)_{i=1}^m$ и функцией ошибки $L(y,a)$. Предположим, мы уже построили алгоритм a, теперь строим алгоритм b таким образом, чтобы

$$\sum_{i=1}^{m} L(y_i, a(x_i) + b(x_i)) \to \min, \qquad \text{(XX.3)}$$

т.е. в идеале мы хотим выполнения равенств

$$a(x_i) + b(x_i) = y_i,\ i \in \{1, 2, \ldots, m\}$$

(с точностью до ошибок алгоритма b). Такой алгоритм b осуществляет поправку ответов алгоритма a до верных ответов на обучающей выборке, т.е. на **невязку (residual)** $\varepsilon_i = y_i - a(x_i)$. Вообще говоря, нельзя считать, что мы получили новую задачу с обучающей выборкой $(x_i, \varepsilon_i)_{i=1}^m$ и функцией ошибки $L(y, a)$, поскольку наша задача оптимизации (XX.3) может отличаться от задачи

$$\sum_{i=1}^{m} L(y_i - a(x_i), b(x_i)) \to \min \qquad \text{(XX.4)}$$

(справедливости ради, заметим, что для большинства разумных функций ошибок эти задачи эквивалентны).

Если описанную идею применить рекурсивно, получится следующий алгоритм:

0. Начать с $a_0(x) \equiv 0$
1. Цикл по $k = 1, \ldots, n$

$$(b, \eta) = \arg\min_{b,\eta} \sum_{i=1}^{m} L(y_i, a_{k-1}(x_i) + \eta b(x_i))$$

$$a_k = a_{k-1} + \eta b$$

Итоговый алгоритм: a_n.

Пример. В **L2–бустинге** используется функция ошибки MSE $L(y, a) = (y - a)^2$, пусть также $\eta = 1$, тогда построение «добавочного» алгоритма b сводится к такой задаче оптимизации

$$\sum_{i=1}^{m} (y_i - a_{k-1}(x_i) - b(x_i))^2 \to \min$$

и тут фактически решается обычная регрессионная задача с метками $y_i - a_{k-1}(x_i)$, т.е. мы настраиваемся на невязки.

Особенность терминологии, когда говорят о бустинге следующая. Базовые алгоритмы называются **«слабыми» (weak learners),** а ансамбль – **«сильным»**

алгоритмом (strong learner). Сам термин бустинг (boosting) переводится как «усиление», т.е. слабые алгоритмы становятся сильным ансамблем[1].

Опишем первый популярный вариант бустинга – AdaBoost. Сейчас из-за большей универсальности и эффективности его практически вытеснил градиентный бустинг. Тем не менее, в его описании есть несколько интересных особенностей, которые полезны с методической точки зрения.

Пусть решается задача бинарной классификации: $Y = \{\pm 1\}$, базовые классификаторы b также будут генерировать метки $b(x) \in \{\pm 1\}$, ансамбль строим в виде

$$a(x) = \text{sgn}\left(\sum_{j=1}^{n} \alpha_j b_j(x) \right),$$

что представляется логичным: коэффициент α_j отражает степень доверия j-му базовому алгоритму. Мы будем использовать суррогатную функцию ошибки и оценивать 0-1-loss сверху экспоненциальной функцией ошибки (exponential loss):

$$L(y,a) = \text{exploss}(y,a) = \exp\left(-y \sum_{j=1}^{n} \alpha_j b_j(x) \right),$$

см. рис. XX.6, $I[y_i \neq a(x_i)] \leq \text{exploss}(y_i, a(x_i))$. Это очень естественная функция: наша задача в том, чтобы линейная комбинация была «правильного знака», ошибка всегда ненулевая, но при угадывании знака она стремится к нулю при увеличении модуля линейной комбинации, а при неугадывании – резко возрастает (при увеличении того же модуля).

Далее также мы будем рассматривать некоторое вероятностное распределение, заданное на объектах обучающей выборки: $W = (w_1, \ldots, w_m) \geq 0$, каждому объекту x_i приписан вес (вероятность) w_i, $w_i \geq 0$ (на самом деле даже $w_i > 0$),

$$\sum_i w_i = 1.$$

Для такого распределения введём **ошибку, порождённую распределением**, как сумму весов объектов, на которых алгоритм ошибся:

[1] Далее мы приведём теоремы, в которой это утверждается. Терминология связана с теорией обучения (learning theory): https://en.wikipedia.org/wiki/Probably_approximately_correct_learning

$$e_W(a) = \sum_{t:a(x_t)\neq y(x_t)} w_t = \sum_{t=1}^{m} w_t I[a(x_t) \neq y(x_t)].$$

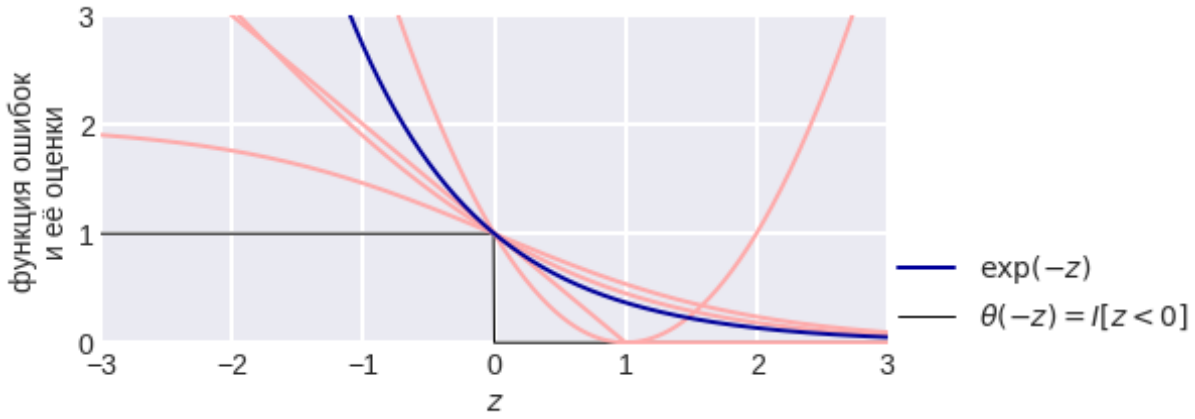

Рис. XX.6. Иллюстрация функции ошибки exponential loss.

Будем строить ансамбль в виде

$$a_n(x) = \text{sgn}[\alpha_1 b_1(x) + \ldots + \alpha_n b_n(x)],$$

последовательно добавляя базовые алгоритмы $b_1(x),\ldots,b_n(x)$ с соответствующими коэффициентами. Теперь запишем псевдокод нашего алгоритма, который перевзвешивает выборку (назначая большие веса объектам, на которых ошибались ранее), обучает новый базовый **слабый (week)** классификатор на взвешенной выборке и добавляет его в ансамбль. Сама процедура уменьшает смещение[1], т.к. фокусируется на «плохо классифицированных» объектах.

AdaBoost: перевзвешивание выборки (чем больше ошибок на объекте раньше, тем больше вес), обучение слабого классификатора на взвешенной выборке и добавление его в ансамбль.

Алгоритм AdaBoost

0. Пусть начальное вероятностное распределение

$$W = (w_1,\ldots,w_m),\ w_i = 1/m,\ i = 1,\ldots,m.$$

1. Цикл по $j = 1,\ldots,n$

1.1. Строим классификатор b_j, минимизируя вероятность ошибки $e_W(b_j)$ в соответствии с текущим распределением:

[1] Подробнее про смещение см. в главе «Сложность».

$$e_W(b_j) = \sum_{t:b_j(x_t)\neq y(x_t)} w_t \,,$$

логично предполагать[1], что $0 < e_W(b_j) < 0.5$.

1.2. Пусть

$$\alpha_j = \frac{1}{2}\ln\left(\frac{1-e_W(b_j)}{e_W(b_j)}\right), \qquad \text{(XX.5)}$$

перестроим распределение – делаем перевзвешивание, увеличивая веса объектов, на которых ошиблись:

$$w_t \leftarrow \frac{w_t \exp(-\alpha_j y(x_t) b_j(x_t))}{\sum_{i=1}^{m} w_i \exp(-\alpha_j y(x_i) b_j(x_i))} \qquad \text{(XX.6)}$$

(знаменатель нужен, чтобы интерпретировать вектор весов как распределение вероятности).

3. Итоговый ансамбль: $a_n(x) = \text{sgn}[\alpha_1 b_1(x) + \ldots + \alpha_n b_n(x)]$.

Теперь опишем, откуда взялись формулы (XX.5)-(XX.6) и какими свойствами будет обладать построенный ансамбль. Число ошибок ансамбля a_s в соответствии с выбранной суррогатной функцией ошибки оценим как

$$\sum_{t=1}^{m} I[y_t \neq a(x_t)] \leq \sum_{t=1}^{m} L(y_t, a(x_t)) = \sum_{t=1}^{m} \exp\left(-y_t \sum_{j=1}^{s-1} \alpha_j b_j(x_t) - y_t \alpha_s b_s(x_t)\right) =$$

$$= \sum_{t=1}^{m} \exp\left(-y_t \sum_{j=1}^{s-1} \alpha_j b_j(x_t)\right) \exp\left(-y_t \alpha_s b_s(x_t)\right) \sim$$

$$\sim \sum_{t=1}^{m} w_t \exp\left(-y_t \alpha_s b_s(x_t)\right) \qquad \text{(XX.7)}$$

[1] Если ошибка равна нулю, то при ненулевых весах это означает, что мы верно классифицируем выборку этим базовым алгоритмом. Если ошибка не меньше 0.5, то построенный алгоритм не лучше случайного. Кроме того, если ошибка больше 0.5, то взяв ответы алгоритма со знаком минус («инвертировав его»), мы получаем алгоритм с приемлемой ошибкой.

Волна (~) здесь означает пропорциональность, вес w_t равен выделенному синим выражению с точностью до знаменателя в (XX.6). Распишем, как менялись веса в AdaBoost (с точностью до нормирующего знаменателя) в соответствии с формулой (XX.6):

$$w_t \sim \frac{1}{m} \cdot \exp(-\alpha_1 y(x_t) b_1(x_t)) \cdot \exp(-\alpha_2 y(x_t) b_2(x_t)) \cdot \ldots \cdot \exp(-\alpha_{s-1} y(x_t) b_{s-1}(x_t)),$$

таким образом

$$w_t \sim \exp(-\alpha_1 y(x_t) b_1(x_t) - \alpha_2 y(x_t) b_2(x_t) - \alpha_{s-1} y(x_t) b_{s-1}(x_t)) =$$

$$= \exp(-y(x_t)(\alpha_1 b_1(x_t) + \alpha_2 b_2(x_t) + \ldots + \alpha_{s-1} b_{s-1}(x_t))) = \text{exploss}(y(x_t), a_{s-1}(x_t))$$

и вес объекта на текущей итерации с точностью до нормировки соответствует экспоненциальной ошибке текущего ансамбля на этом объекте. Применив формулу (XX.6), мы просто пересчитываем exploss с учётом модификации ансамбля:

$$\sum_{t=1}^{m} I[y_t \neq a(x_t)] \leq \sum_{t=1}^{m} L(y_t, a(x_t)) \sim \sum_{t=1}^{m} \underbrace{w_t \exp\left(-y_t \alpha_s b_s(x_t)\right)}_{\substack{\text{exploss на s-1} \\ \text{exploss на } s}}. \qquad \text{(XX.7')}$$

Это ещё раз переписанная формула (XY.7). Фактически, мы обосновывали применение формулы (XX.6): она позволяет формировать веса пропорциональными экспоненциальной ошибке.

Рассмотрим выражение (XX.7), желательно его минимизировать, сделаем преобразования:

$$\sum_{t=1}^{m} w_t \exp\left(-y_t \alpha_s b_s(x_t)\right) =$$

$$= \sum_{t: y_t = b_s(x_t)} w_t \exp\left(-\alpha_s\right) + \sum_{t: y_t \neq b_s(x_t)} w_t \exp\left(\alpha_s\right) =$$

$$= (1 - e_W) \exp\left(-\alpha_s\right) + e_W \exp\left(\alpha_s\right). \qquad \text{(XX.8)}$$

Здесь одна из сумм в точности совпадает с ошибкой, порождённой распределением, а сумма сумм равняется 1, отсюда получаем итоговую ошибку. Для её минимизации по α_s возьмём производную и приравняем к нулю:

$$-(1-e_W)\exp(-\alpha_s)+e_W\exp(\alpha_s)=0.$$

После преобразования получим формулу (XX.5), т.е. по известной ошибке e базового алгоритма можно найти оптимальный для него коэффициент в линейной комбинации:

$$\alpha_s=\frac{1}{2}\log\frac{1-e}{e}.$$

График $\alpha_s(e)$ показан на рис. XX.7: для малой ошибки вес очень большой, при $e=0.5$ вес обращается в ноль, при $e>0.5$ вес отрицательный (что довольно логично).

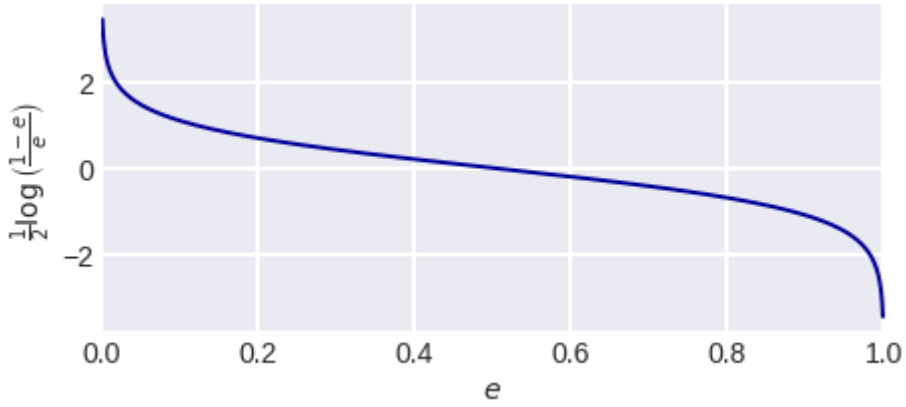

Рис. XX.7. Зависимость веса от ошибки.

Если подставить найденный коэффициент в формулу (XX.8), то получаем (опускаем индекс W)

$$\propto(1-e)\exp\left(-\log\sqrt{\frac{1-e}{e}}\right)+e\exp\left(\log\sqrt{\frac{1-e}{e}}\right)=$$

$$=\frac{(1-e)\sqrt{e}}{\sqrt{1-e}}+\frac{e\sqrt{1-e}}{\sqrt{e}}=2\sqrt{e(1-e)}\le\exp(-2(0.5-e)^2).$$

Теорема. Если на каждом шаге мы можем построить слабый (weak) классификатор с ошибкой $e_W(b_j)\le 0.5-\varepsilon$, где $\varepsilon>0$, то на обучающей выборке ошибка всего ансамбля

$$a(x)=\operatorname{sgn}\left(\sum_{j=1}^{n}\alpha_j b_j(x)\right)$$

оценивается как

$$\sum_{t=1}^{m} I[a(x_t) \neq y(x_t)] \leq \exp(-2\varepsilon^2 n).$$

Доказательство. Пусть для промежуточного ансамбля a_s

$$\sum_{t=1}^{m} I[y_t \neq a_s(x_t)] \leq \sum_{t=1}^{m} L(y_t, a_s(x_t)) = \sum_{t=1}^{m} \exp\left(-y_t \sum_{j=1}^{s} \alpha_j b_j(x_t)\right) \equiv Z_s,$$

получим оценку на это выражение, представив

$$Z_n = \frac{Z_1}{Z_0}\frac{Z_2}{Z_1} \cdot \ldots \cdot \frac{Z_n}{Z_{n-1}},$$

если верно $Z_s / Z_{s-1} \leq \exp(-2\varepsilon^2)$, то утверждение доказано. Действительно,

$$\frac{Z_s}{Z_{s-1}} = \frac{\sum_{t=1}^{m} \exp\left(-y_t \sum_{j=1}^{s-1} \alpha_j b_j(x_t) - y_t \alpha_s b_s(x_t)\right)}{\sum_{t=1}^{m} \exp\left(-y_t \sum_{j=1}^{s-1} \alpha_j b_j(x_t)\right)} = \sum_{t=1}^{m} w_t \exp\left(-y_t \alpha_s b_s(x_t)\right).$$

Для последнего выражения уже доказали оценку $\leq \exp(-2(0.5 - e_W)^2)$. Теорема доказана.

Таким образом, всего лишь из предположения, что слабые классификаторы «на ε лучше случайного», следует возможность объединения их в сколь угодно точный ансамбль (ошибка уменьшается экспоненциально от числа слабых алгоритмов ансамбля). Однако заметим, что мы должны **для любого распределения W уметь строить такой слабый классификатор**, что является не очень слабым требованием.

На рис XX.8. показаны итерации AdaBoost, размер объектов пропорционален их весам. Видно, что веса объектов, на которых происходит ошибка увеличиваются на следующей итерации. В качестве базовых алгоритмов взяты пеньки (stumps): деревья глубины 1. С точки зрения геометрии решения базовый алгоритм пытается разделить объекты вертикальной или горизонтальной прямой. Первый базовый алгоритм довольно неплохо разделил объекты разных классов обучающей выборки, но совершил 4 ошибки. Поэтому у следующего алгоритма веса 4х объектов повышены, он 3 из них классифицировал верно. Следующему базовому алгоритму достался совсем большой вес

Возможность «усиления» не всегда очевидна!

одного из объектов (синего), т.к. на нём ошибались все его предшественники. Он уже классифицировал его верно, хотя его разделение выборки не кажется логичным (слишком много ошибок на остальных объектах, но их веса малы – их верно классифицируют предыдущие базовые алгоритмы).

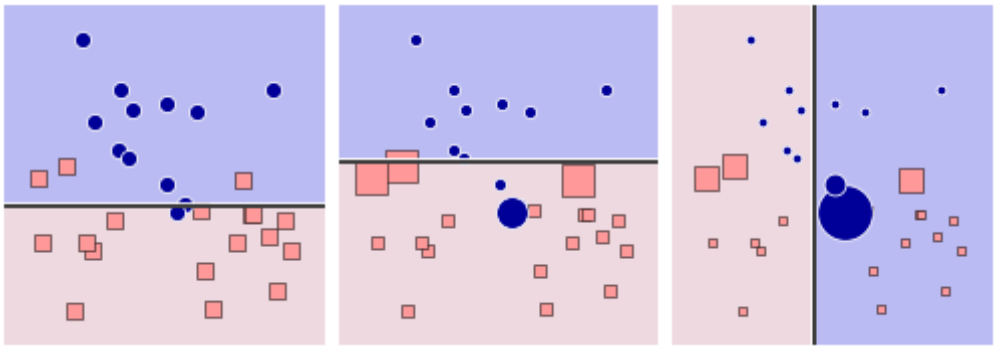

Рис. XX.8. Итерации AdaBoost: (1я, 2я и 3я).

На рис. XX.9 показано итоговое решение (через некоторое число итераций), полученное с помощью AdaBoost. Оно имеет уже более сложную разделяющую поверхность по сравнению с базовыми пеньками, однако «проблемный» для классификации синий объект так и не был верно классифицирован[1].

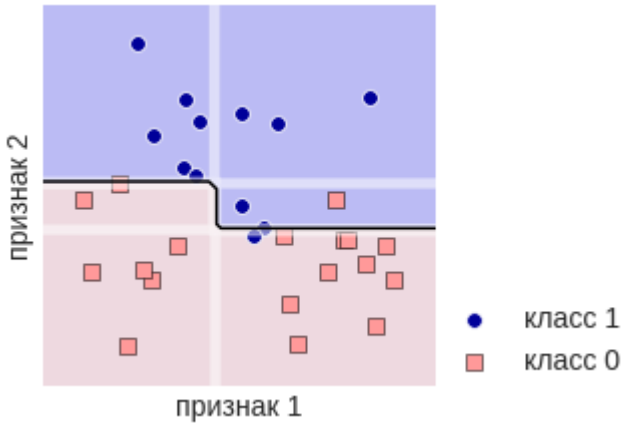

Рис. XX.9. Итоговое решение с помощью AdaBoost.

Если алгоритмы базовой модели допускают минимизацию весовой ошибки классификации для произвольного распределения, то построение AdaBoost с помощью **перевзвешивания** (**re-weighting,** т.е. применения формулы (XX.6)) не представляется затруднительным. В противном случае применяют технику **«пересэмплирования»** (**re-sampling**) – для обучения очередного базового классификатора берут подвыборку обучающей выборки. Подвыборка берётся с

[1] В списке вопросов будет задание «заставить его верно классифицироваться».

возвращением, такого же объёма, как исходная выборка, на каждой итерации t-й объект выбирается с вероятностью пропорциональной значению w_t.

Отметим, что с каждой итерацией очередной базовый алгоритм всё больше «затачивается» на правильную классификацию именно тех объектов, на которых чаще ошибались предыдущие базовые алгоритмы ансамбля. Если в задаче много выбросов, то это приводит к ухудшению обобщающей способности: ансамбль начинает «настраиваться на шум». Веса объектов при построении AdaBoost можно как раз использовать для идентификации выбросов.

Склонность к переобучению заложена в саму схему бустинга.

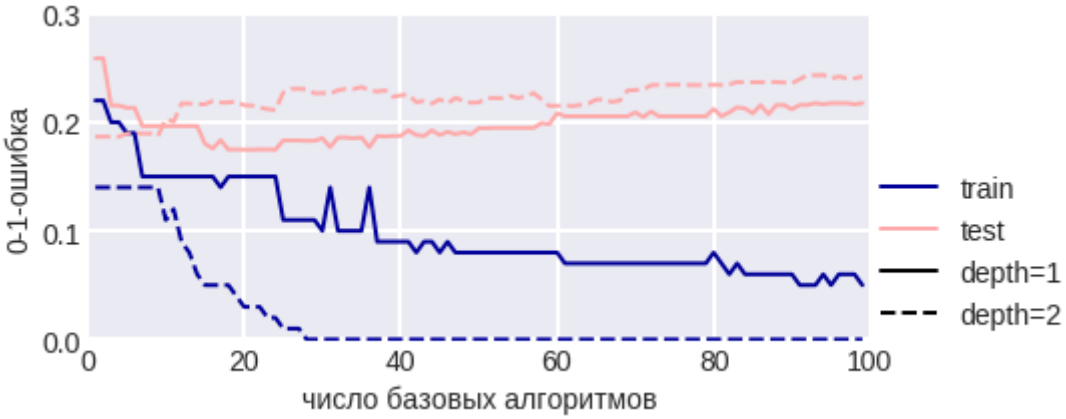

Рис. XX.10. Уменьшение ошибки AdaBoost для деревьев разной глубины.

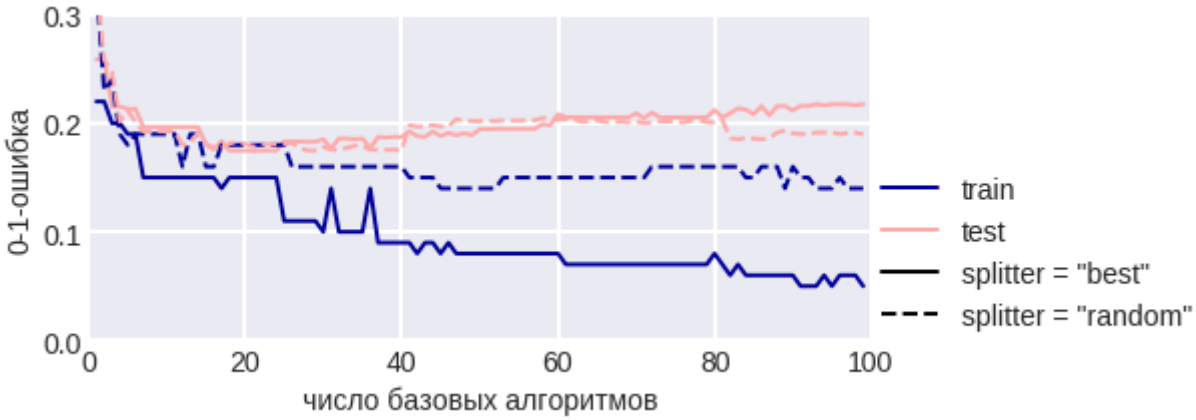

Рис. XX.11. Уменьшение ошибки AdaBoost для деревьев с разными стратегиями расщеплений.

На рис. XX.10-11 показано, как уменьшается ошибка ансамбля AdaBoost при увеличении размера ансамбля (числа алгоритмов в нём). На рис. XX.10 показаны графики ошибки на обучающей и тестовой выборках для AdaBoost при использовании в качестве базовых алгоритмов деревьев глубины 1 и 2. Эффект переобучения виден на всех графиках (разрыв в качестве на обучении и тесте), но для более сложной модели (ансамбля над более глубокими деревьями) он больше. При этом более сложный ансамбль смог достичь

нулевой ошибки на обучении. На рис. XX.11 в качестве базовых алгоритмов выбраны деревья малой глубины с оптимальными расщеплениями и со случайными расщеплениями. В целом, у них сравнимое качество на тесте, но у последних меньше эффект переобучения: случайные расщепления играют роль регуляризатора (усложняют подгонку под обучающие данные).

Интересно, что в бустинге ошибка на тесте может убывать даже при достижении нулевой ошибки на обучении. Есть теория, которая связывает бустинг с максимизацией зазора (margin), а максимизацию зазора с улучшением обобщения.

Популярный градиентный бустинг рассмотрим в отдельной главе.

Деформация целевого признака, многоклассовая классификация

Как обсуждалось выше, для повышения разнообразия базовых алгоритмов варьируют (меняют) некоторые сущности. Например, можно варьировать целевой вектор $y=(y_1,\ldots,y_m)$, подвергая его деформации $\varphi:\mathbb{R}\to\mathbb{R}$, будем использовать обозначение $\varphi(y)=(\varphi(y_1),\ldots,\varphi(y_m))$ (деформация действует поэлементно на вектор) и предполагать, что деформация обратима, т.е. существует φ^{-1}. Для неотрицательных целевых значений часто используют деформации вида: $\varphi(y)=\log(y+1)$, $\varphi(y)=y^p$.

Если задаться некоторым набором деформаций $\varphi_1,\ldots,\varphi_n$, то возможно построение ансамбля следующим образом

1. Цикл $t=1,\ldots,n$.

    1.1. Учим базовый алгоритм на деформированных целевых значениях:

$$b_t=\text{fit}(X,\varphi_t(y)).$$

2. Итоговый ансамбль (в задаче регрессии) получает метку объекта x по формуле:

$$a(x)=\frac{1}{n}\sum_{t=1}^{n}\varphi_t^{-1}(b_t(x)).$$

Здесь не надо забыть подвергнуть обратной деформации ответы каждого базового алгоритма b_t. Недостаток у такого ансамбля в том, что обычно обучение каждого базового алгоритма (fit) происходит с определённой функцией ошибки, после обратной деформации нет гарантий, что алгоритм $\varphi_t^{-1}(b_t(\cdot))$, тем более $a(\cdot)$, минимизирует нужную функцию ошибки. Впрочем, иногда удаётся подобрать деформацию, чтобы это выполнялось (подробнее в главе о решающих правилах). Например, в задаче регрессии с помощью решающих деревьев можно использовать деформации вида

$$\varphi(y) = y + f_j(x), \qquad \text{(XX.9)}$$

где f_j – значение j-го признака. Это не вызовет проблем с минимизацией MSE.

Поговорим теперь о **многоклассовой классификации с помощью бинарных классификаторов**. Предположим, что мы решаем задачу классификации с l непересекающимися классами[1], при этом решать мы её хотим бинарными классификаторами. Первый стандартный подход – **One-vs-All (One versus All / the rest)** – последовательно **отделять каждый класс от остальных**, т.е. построить классификаторы «объект первого класса / не первого», «объект второго класса / не второго» и т.д. Фактически мы проводим такое кодирование значений целевого вектора:

$$0 \to 1000$$
$$1 \to 0100$$
$$2 \to 0010$$
$$3 \to 0001$$

(здесь и ниже приводим примеры для $l = 4$ классов) и решаем l бинарных задач классификации, каждая соответствует столбцу в единичной матрице кодировки.

1. Цикл $t = 1,\ldots,l$.

    1.1. Учим базовый алгоритм на перекодированных целевых значениях:

$$b_t = \text{fit}(X, \varphi_t(y)),$$

[1] Если классы пересекаются, тогда просто строим l классификаторов (по числу классов), каждый предсказывает, лежит ли объект в соответствующем классе.

где $\varphi_t(y) = I[y = t]$.

2. Для классификации объекта x получаем оценку вероятности принадлежности t-му классу $p_t = b_t(x)$ и выбираем максимальную оценку:

$$a(x) = \arg\max_t p_t .$$

Второй подход – **One-vs-One (каждый против каждого)** – для всех пар классов (i, j): $1 \le i < j \le l$ строим классификатор b_{ij}, который отделяет объекты i-го класса от объектов j-го класса. Это соответствует такой кодировке:

$$0 \to 111{-}{-}{-}$$
$$1 \to 0{-}{-}11{-}$$
$$2 \to {-}0{-}0{-}1$$
$$3 \to {-}{-}0{-}00$$

Здесь прочерк означает, что мы не кодируем это значение (объекты данного класса не будут участвовать в обучении соответствующего бинарного классификатора). Пусть алгоритм b_{ij} для объекта x получает две вероятности: p_{ij} – вероятность принадлежности к классу i, p_{ji} – вероятность принадлежности к классу j, связанные формулой

$$p_{ij} = 1 - p_{ji} .$$

Итоговый ансамбль можно строить, например, следующим образом:

$$a(x) = \arg\max_i \sum_{j \ne i} p_{ij} .$$

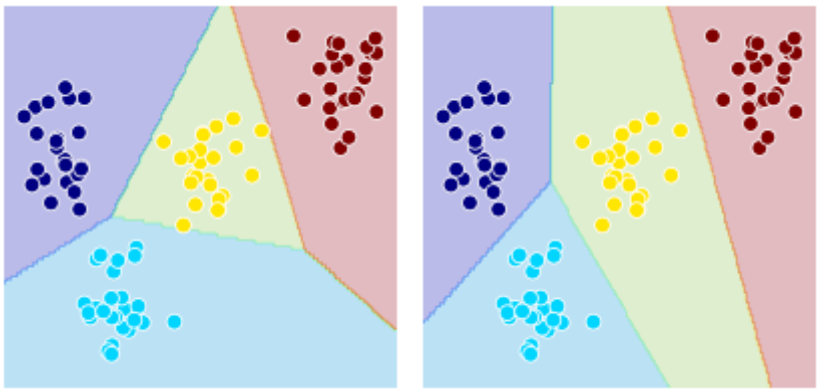

Рис. XX.12. Классификация с помощью логистических регрессий подходами One versus All (слева) и One versus One (справа).

Как нетрудно видеть, можно использовать произвольную кодировку, например

$$0 \to 00$$
$$1 \to 01$$
$$2 \to 10$$
$$3 \to 11$$

Здесь используется минимально возможное число бинарных классификаторов (это т.н. «экономное кодирование»). Удобство такой кодировки в том, что ответы двух бинарных классификаторов представляют двоичную форму записи номера класса, к которому следует отнести объект. Проблема такой кодировки в том, что если один из классификаторов ошибся, то мы сразу же получаем неверный ответ. Если же матрица кодирования будет составлена из вектор-строк, которые находятся на расстоянии Хэмминга[1] не меньше 3 друг от друга, то после ошибки любого классификатора на объекте (если остальные не ошиблись), мы всё-таки сможем правильно классифицировать объект. Именно такие кодировки и получают коды, исправляющие ошибки. Пример такой кодировки:

$$0 \to 000111$$
$$1 \to 011100$$
$$2 \to 101010$$
$$3 \to 110001$$

Здесь придётся обучить 6 бинарных классификаторов – по числу столбцов в бинарной матрице кодировки, каждый решает задачу с кодировкой, соответствующей его столбцу. Так например, первый пытается отделить объекты классов 0 и 1 от объектов классов 2 и 3.

ECOC-ансамбли (Error-Correcting Output Code»[2]) как раз используют коды, корректирующие ошибки. Эти и другие ансамбли, использующие кодировки, можно записать следующим образом[3].

0. Строится бинарная $m \times k$-матрица кодировки $H = \| h_{ij} \|$, i-ю строку которой обозначим за $H[i,:]$, j-й столбец – за $H[:, j]$.

[1] Расстояние между векторами равно числу координат, в которых они различаются.

[2] Dietterich T. G., Bakiri G. Solving multiclass learning problems via error-correcting output codes //Journal of artificial intelligence research. – 1994. – Т. 2. – С. 263-286.

[3] Это, по сути, стекинг с мета-алгоритмом «ближайший сосед», см. главу про стекинг.

1. Цикл $t = 1,\ldots,k$.

1.1. Учим базовый алгоритм на перекодированных целевых значениях:

$$b_t = \text{fit}(X, H[:,t]).$$

2. Для классификации объекта x смотрим, на какую строку максимально похож вектор ответов базовых алгоритмов:

$$a(x) = \arg\min_i \rho((b_1(x),\ldots,b_k(x)), H[i,:]).$$

Здесь ρ – заранее выбранная метрика. Если базовые бинарные классификаторы выдают просто метку: 0 или 1, то логично использовать метрику Хэмминга. В примере выше, если надо классифицировать объект, мы запускаем 6 обученных классификаторов, их ответы записываем в виде бинарного вектора, например, (1,0,1,0,0,0), смотрим, к какой строке матрицы кодировки он ближе по расстоянию Хэмминга. В данном случае – к строке, соответствующей классу 2.

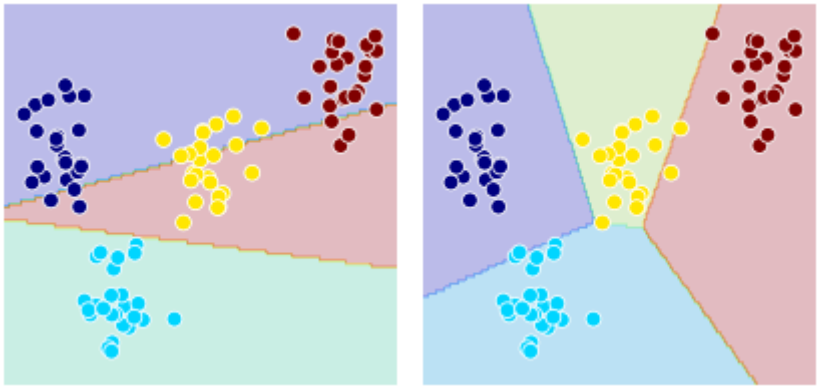

Рис. XX.13. Классификация с помощью логистических регрессий кодированием: экономным (слева) и неэкономным (справа).

Подведём итог, в виде таблицы

метод	**длина ансамбля**	**комментарий**
One-vs-All	l	+ интерпретируемые базовые алгоритмы – некоторые классы сложно отделить от остальных
One-vs-One	C_l^2	+ как правило, классы хорошо отделяются друг от друга – может быть много базовых алгоритмов

Экономное кодирование	$]\log_2 l[$	+ мало базовых алгоритмов – часто алгоритмы не интерпретируемы
ECOC	зависит от используемого кода	+ математическая база из теории кодирования – нет ожидаемого прироста качества

Основная проблема при использовании кодировок в том, что базовые алгоритмы будут произвольное множество классов отделять от остальных, хотя на классах может быть некоторый порядок. Это мешает интерпретации получаемых алгоритмов и получению высокого качества при их обучении. Пусть, например, метки 0, 1, 2, 3 это оценки студента в 4-балльной шкале, которые нужно спрогнозировать. В примерах выше первый базовый алгоритм для экономной и ECOC-кодировки будет отделять объекты с метками 0, 1 от объектов с метками 2, 3, что довольно логично: «слабые работы / слабых студентов» от «сильных». Можно ожидать, что получится его хорошо обучить. Но вот второй алгоритм пытается отделить объекты с чётными метками 0, 2 от объектов с нечётными: 1, 3 (не понятна интерпретация и не стоит ожидать высокого качества). На рис. XX.12-13 показаны примеры использования разных подходов, и видно, когда возникают нелогичные разделения.

Часто если понятно «что делает алгоритм», то удаётся его хорошо обучить.

Использование кодов, исправляющих ошибки, с одной стороны, является примером привлечения красивой математики из смежной области Computer Science: теории кодирования. С другой стороны, как мы видим, теряется интерпретация. Кроме того, в теории кодирования ошибки в каждом бите предполагаются равновероятными, в машинном обучении разные базовые алгоритмы будут иметь, вообще говоря, разную вероятность ошибки.

Напомним также, что для логистической регрессии предлагался другой способ решения задачи классификации на несколько классов (тоже, по сути, ансамблевый):

$$a(x) = \text{softmax}(b_1(x),\ldots,b_k(x)).$$

Приложения ансамблей

1. Как мы видели, разные подходы в ансамблировании тесно связаны. Скажем, стекинг (см. главу про стекинг) является самой общей формой ансамблирования, а его частные случаи: усреднения, ансамбли на кодировках и

т.п. Чаще используют простейшую форму ансамблирования: усреднение базовых алгоритмов. Кроме того, есть т.н. «ручные способы ансамблирования, например в одном из соревнований по анализу данных[1] для ансамблирования двух алгоритмов использовалась мета-функция $f(a_1, a_2)$, заданная таблицей

	$a_1 \le 0.1$	$0.1 < a_1 < 0.9$	$a_1 \ge 0.9$
$a_2 \le 0.1$	$\min(a_1, a_2)$	$\min(a_1, a_2)$	$0.55a_1 + 0.45a_2$
$0.1 < a_2 < 0.9$	$0.1a_1 + 0.9a_2$	$\text{mean}(a_1, a_2)$	$0.9a_1 + 0.1a_2$
$a_2 \ge 0.9$	$0.75a_1 + 0.25a_2$	$\max(a_1, a_2)$	$\max(a_1, a_2)$

2. Идеи варьирования можно активно использовать. На практике разные признаки могут быть

- получены из разных источников (скажем, в скоринге – из разных БКИ),
- разной природы (по типу признака: вещественные, категориальные и т.д.).

Напрашивается разделить признаки на группы и обучать базовые алгоритмы, используя разные комбинации групп. Также одни и те же признаки можно предварительно подвергать разным преобразованиям (кодированию, логарифмированию и т.п.). Поэтому при построении базовых алгоритмов варьируют и преобразования. Аналогично, объекты также можно разделить на группы (особенно, если они упорядочены по времени: группами можно считать недели, месяцы и т.п.)

3. На соревнованиях по машинному обучению часто одна модель существенно превосходит остальные по качеству. При этом качество ещё больше повышают ансамблированием. Для нейросетей используют CV-ансамбли (при их обучении необходимо следить за качеством на отложенной выборке, поэтому логично использовать схемы с варьированием обучающей выборки), для бустинга усредняют алгоритмы с разными значениями гиперпараметров (или взятые в разных библиотеках – они отличаются деталями реализации). Хорошая практика – поиск оптимальной линейной комбинации случайных лесов с разным значением гиперпараметра max_features (число признаков, которые просматриваются при выборе расщепления).

[1] Данный способ применил Дмитрий Ефимов в соревновании «Amazon Employee Access Challenge».

Вопросы и задачи

1. При обосновании эффективности усреднения регрессоров требовалось равенство дисперсий базовых регрессоров на конкретном объекте. Сделайте обоснование, отказавшись от равенства: пусть у всех регрессоров будет конечная дисперсия. Сделайте обоснование, отказавшись от независимости регрессоров.

2. В главе была использована фраза «ответы всех хороших регрессоров, скорее всего, коррелируют с целевыми значениями». Можно привести пример хороших регрессоров (в смысле разумности их использования на практике), ответы которых плохо коррелируют с целевыми значениями на обучающей выборке?

3. В главе был пример того, что разнообразные базовые алгоритмы позволяют получить более качественный комитет большинства. Можно ли построить сходный по качеству комитет из однообразных алгоритмов?

4. В примерах бустинга / бэгинга были объекты, которые в итоге неверно классифицированы ансамблем. Верно ли, что при возможности варьировать число базовых алгоритмов бустинг всегда настраивается на обучающую выборку (скажем, в задаче классификации будет 100%-я точность)? Аналогичный вопрос для бэгинга.

5. Была приведена деформация (XX.9) для ансамбля над решающими деревьями. Для каких ещё базовых алгоритмов она подойдёт? Какие ещё деформации, использующие признаки, можно предложить?

6. Для каких функций ошибки задачи (XX.3) и (XX.4) эквивалентны?

7. Можно ли привести пример задачи и модели алгоритмов, при которых не возможен бустинг (т.е. усиление): нельзя гарантировать выполнение условий теоремы об адаптивном бустинге.

Ансамбли: итоги

Существуют следующие основные модели ансамблирования:

модель или общая идея	описание и примеры
комитеты (голосование) / усреднение	Построение разнообразных (в идеале – независимых) алгоритмов и их усреднение / голосование по ним, в том числе с помощью **бэгинга** и предварительной деформации ответов. Здесь же **случайный лес**.
кодировки / деформации ответов	Специальные преобразования или кодировки целевых значений, сведение решения задачи к решению нескольких задач. Пример: **ECOC-ансамбли**.
стекинг (stacking)	построение метапризнаков – ответов базовых алгоритмов на объектах выборки, обучение на них мета-алгоритма
бустинг	Построение суммы нескольких алгоритмов. Каждое следующее слагаемое строится с учётом ошибок предыдущих. Примеры: **адаптивный**, **градиентный бустинг**.
«ручные методы»	Эвристические способы комбинирования ответов базовых алгоритмов (с помощью визуализаций, обучения в специальных подпространствах и т.п.)
однородные ансамбли	Алгоритм по построению является ансамблем и состоит из однородных частей. Пример: **нейронные сети**[1].

[1] Здесь нейронные сети не обсуждались.

Спасибо за внимание к книге!
Замечания по содержанию, замеченные ошибки
и неточности можно написать в телеграм-чате
https://t.me/Dyakonovsbook